WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Monday, July 26, 2004

Hide?	Set Name	Query	Hit Count
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
☐	L13	L12 same (stor$4 near2 request)	6
☐	L12	L10 with queue	14
☐	L11	L10 same queue	25
☐	L10	(plurality or multiple) near2 (shift adj register)	6645
☐	L9	L8 same shift$4	1
☐	L8	l3 same queue	68
	DB=USPT; PLUR=YES; OP=OR		
☐	L7	US-5170483-A.did.	1
☐	L6	US-5170483-A.did.	1
	DB=USPT,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
☐	L5	l3 same (shift adj register)	2
☐	L4	l1 and L3	0
☐	L3	(maintain$4 near3 request near3 order)	153
☐	L2	L1 same (queue adj structure)	1
☐	L1	(index adj shifter)	51

END OF SEARCH HISTORY

First Hit Previous Doc Next Doc Go to Doc#

Generate Collection Print

L13: Entry 2 of 6 File: PGPB Sep 19, 2002

DOCUMENT-IDENTIFIER: US 20020133658 A1
TITLE: Method of synchronizing arbiters within a hierarchical computer system

Detail Description Paragraph:
[0036] FIG. 5 presents a block diagram of an L2 port. When the L2 port receives a transaction from a CPU port via a CPU-L1 bus, the transaction passes through input multiplexer 505. The transaction is then passed to the L1-L2 bus. The transaction is also stored in an outgoing request queue (ORQ) 510. The ORQ 510 may be a plurality of registers, such as shift registers or may be a buffer, such as a first-in-first-out buffer, a circular buffer, or a queue buffer. The ORQ 510 may be any width sufficient to store transactions. In one embodiment of the invention, the ORQ 510 is 62 bits wide and contains storage for 16 transactions. The 2 extra bits may be utilized to store a transaction and information that identifies which of the three CPU ports originated the transaction. In addition, other methods known by those skilled in the art may be utilized to indicate the origin of a transaction.

Previous Doc Next Doc Go to Doc#

L5: Entry 1 of 2 File: TDBD May 1, 1991

TDB-ACC-NO: NN9105293

DISCLOSURE TITLE: Processor/Memory Switch Which Maintains the Temporal Ordering of Requests.

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, May 1991, US

VOLUME NUMBER: 33
ISSUE NUMBER: 12
PAGE NUMBER: 293 - 296

PUBLICATION-DATE: May 1, 1991 (19910501)

CROSS REFERENCE: 0018-8689-33-12-293

DISCLOSURE TEXT:

- To alleviate performance problems associated with slow memories (relative to processor cycle times), most large computer systems rely on "banking" to create the appearance of a faster memory. Typical banking schemes partition physical memory into N identical blocks, with consecutive addresses in adjacent blocks; N consecutive requests can be made before returning to the original bank. In this case, the apparent memory cycle time becomes the actual memory cycle time divided by the number of banks, N. The resulting gain in performance is at the expense of complexity. A processor sending a request to one of N banks needs a switch (or routing network) to steer its request to a given bank. In a multiple-processor system, as the number of processors increase, the rate of requests increase, and the number of banks must correspondingly increase to maintain the same apparent memory cycle time. In high-performance systems, the switch's cost can be significant since the data paths are extremely wide and small switch/memory latencies are one key to a high-performance memory system. To decrease cost with minimal performance degradation, the memory partitioning or "banking" is often hierarchical; banks are grouped together into BSMs (Base Storage Modules). On a given cycle, any set of "ready" banks can be accessed provided that the set does not include two banks from a common module. - Fig. 1 illustrates a typical Prior Art memory subsystem. For each Base Storage Module (BSM) or group of banks, an arbitrator selects (at most) one of the processors' requests destined for the associated BSM. A conflict-free (i.e., cross-bar) switch may be used since the arbitrators establish a one-to-one relationship between requestors and BSMs. Since requests may be destined for banks which are still busy from previous requests, a "BSM queue" is included prior to each BSM. Requests can experience arbitrary delays as a result of the arbitration requirement and the random profile of BSM queue wait times. These arbitrary delays are manifested in a temporal reordering of fetched data as seen by a given requestor. (When a request experiences little delay, it may return before a logically previous request that experienced a substantial delay.) This temporal reordering requires two additional hardware complexities in the return (memory to requestor) path. First, contention can occur between two or more BSMs attempting to return data to a common requestor. This means that an arbitration unit must be placed in the return path. The arbitration unit may block some return packets for a unknown number of cycles, thereby requiring queues for

data coming back from the memory. Temporal reordering in such a design also forces the requestor to provide tags (on fetch requests) and circuitry which inspects the tags of returned data to reconstruct the proper sequence of the fetched data. - In the system shown in Fig. 2, the arbitrator 12 will be forced to select among those requests which are destined for "ready" banks. Therefore, any request selected by the arbitrator 12 (1-M) will be immediately accepted by the memory unit 14 (1-M). In the proposed design, only one request per requestor is allowed to be presented to the arbitration units 12 (1-M). Once the first entry in the requestor queue has been selected by the appropriate arbitration unit 12n, subsequent entries advance one position. Therefore, the collection of arbitration units 12 always dispatches requests from a given requestor 16 (1-R), in the same sequence that the requestor presents them to the switch 18. Since the proposed design also guarantees a fixed amount of delay from the selection of a request (by the arbitrator) to any other phase of a memory operation, the order of requests is maintained throughout the network; therefore, no queues or arbitration are required for a request after it is processed by the arbitrator. - To allow the arbitrator to select only request destined for "ready" banks, some mechanism is required to keep track of memory bank activity. One such mechanism is a set of counters. There would be a one-to-one correspondence between a bank and a counter. For a system with B banks per BSM and C processor cycles in the memory busy cycle, each arbitrator would incorporate a set of B counters, each with sufficient bits to count to C. As the arbitrator selects and sends out a memory request to a particular bank, the associated counter would be loaded with the value of C. Each cycle, all non-zero valued counters would be decremented. At any instant, a counter's value would indicate the number of cycles which remain in the associated bank's busy cycle. The enforcement of selecting only from the set of ready banks can then easily be implemented by using the bank select bits from each request to index the associated counter. If the counter value is non-zero, the request is masked. - A shift register is an alternative mechanism for tracking memory bank activity; it would become more efficient as the number of banks grows with respect to the maximum number of banks which can be busy any particular cycle. (The maximum number of busy banks is the same as the value C if only one request per BSM is allowed.) The shift register would require C entries, each wide enough to encode a bank number (within a BSM), maybe 7 bits. As the arbitrator sends a request to the BSM, the bank-select bits (which determine which bank is just about to become busy) are placed in the leftmost entry of the shift register. - Each cycle the shift register contents are shifted one place to the right; therefore, the shift register will maintain a list of all busy banks. (Although not of importance to the operation, it happens that an entry's position within a shift register also indicates the number of cycles which remain before the associated bank becomes available.) By comparing a request to each of the entries in the shift register, it can be determined if the required bank is busy. Since the entries in the shift registers would be only a few bits wide, a separate copy could be maintained for each requestor or a group of requestors (depending on loading considerations). Fig. 3 illustrates a shift register implementation where each pair of requestors (to a given arbitrator) share a shift register.

Previous Doc Next Doc Go to Doc#